UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/608,039	06/30/2003	Sanjay Ghemawat	0026-0030	7559

44989 7590 07/23/2007
HARRITY SNYDER, LLP
11350 Random Hills Road
SUITE 600
FAIRFAX, VA 22030

EXAMINER	
DAYE, CHELCIE L	

ART UNIT	PAPER NUMBER
2161	

MAIL DATE	DELIVERY MODE
07/23/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

UNITED STATES PATENT AND TRADEMARK OFFICE

ED

Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Application Number: 10/608,039
Filing Date: June 30, 2003
Appellant(s): GHEMAWAT ET AL.

MAILED
JUL 23 2007
Technology Center 2100

Paul A. Harrity

For Appellant

EXAMINER'S ANSWER

This is in response to the appeal brief filed December 13, 2006, appealing from the Office action mailed June 14, 2006.

(1) Real Party in Interest

A statement identifying by name the real party in interest is contained in the brief.

(2) Related Appeals and Interferences

The examiner is not aware of any related appeals, interferences, or judicial proceedings, which will directly affect or be directly affected by or have a bearing on the Board's decision in the pending appeal.

(3) Status of Claims

The statement of the status of claims contained in the brief is correct.

(4) Status of Amendments After Final

Te appellant's statement of the status of amendments after final rejection contained in the brief is correct.

(5) Summary of Claimed Subject Matter

The summary of claimed subject matter contained in the brief is correct.

(6) Grounds of Rejection to be Reviewed on Appeal

The appellant's statement of the grounds of rejection to be reviewed on appeal is correct.

(7) Claims Appendix

The copy of the appealed claims contained in the Appendix to the brief is correct.

(8) Evidence Relied Upon

6209003	Mattis	7-1998
20030182330	Manley	3-2002

"New-Value Logging in Echo Replicated File System", by: Andy Hisgen, Andrew Birrell, Charles Jerian, Timothy Mann, Garret Swart, (June 23, 1993).

(9) Grounds of Rejection

The following ground(s) of rejection are applicable to the appealed claims:

Claim Rejections - 35 USC § 102

The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

Claims 1,2,4-7,10-13, and 20-25 are rejected under 35 U.S.C. 102(b) as being anticipated by Mattis (US Patent No. 6,209,003).

Regarding Claims 1,12, and 13, Mattis discloses a method for deleting one or more of a plurality of files, the files including one or more chunks stored by a plurality of servers, the method comprising:

a master connected to the servers and configured to (column 8, lines 60-65, Mattis):

identifying a file to be deleted (column 21, lines 59-62, Mattis; wherein fragments are referring to files),

renaming the identified file (column 3, lines 16-19, Mattis);

permanently deleting the renamed file (column 16, lines 48-52, Mattis) a predetermined amount of time after renaming the identified file (column 22, lines 43-47, Mattis) as part of a garbage collection process (Fig.8A, column 21, lines 52-55, Mattis);

receiving, from the servers, information concerning chunks stored by the servers (column 8, lines 16-19, Mattis; wherein keys are referring to chunks); and

identifying, to one of the servers, one of the chunks that corresponds to the permanently deleted file (column 33, lines 32-40, Mattis).

Regarding Claim 2, Mattis discloses a method wherein the identifying a file to be deleted includes:

receiving a deletion instruction regarding the file (column 22, lines 31-35, Mattis).

Regarding Claim 4, Mattis discloses a method wherein the predetermined amount of time is a user-configurable amount of time (column 22, lines 14-23, Mattis).

Regarding Claim 5, Mattis discloses a method wherein the user-configurable amount of time differs for different ones of the files (columns 22-23, lines 65-67 and 1-5, respectively, Mattis).

Regarding Claim 6, Mattis discloses a method wherein metadata is associated with the files (column 25, lines 2-8, Mattis); and wherein the permanently deleting the renamed file (column 16, lines 48-52, Mattis) includes erasing the metadata associated with the renamed file (column 25, lines 12-16, Mattis).

Regarding Claim 7, Mattis discloses a method comprising:

deleting, by the one of the servers, the one of the chunks that corresponds to the permanently deleted file (Fig. 9B, columns 35 and 36, lines 65-67 and 1-7, Mattis; wherein the process of modifying is interpreted as changing a file whereas the old file no longer exists, along with the chunks that are associated with the files).

Regarding Claim 10, Mattis discloses a method comprising:

maintaining versions of the chunks (column 17, lines 42-46, Mattis);

identifying a stale chunk based on the versions of the chunks (column 26, lines 15-21, Mattis); and

deleting the stale chunk (column 26, lines 21-22, Mattis).

Regarding Claim 11, Mattis discloses a method wherein metadata is associated with the chunks (column 25, lines 2-8, Mattis); and wherein the deleting the stale chunk (column 26, lines 21-22, Mattis) includes erasing the metadata associated with the stale chunk (column 25, lines 12-16, Mattis).

Regarding Claims 20,22,24,and 25, Mattis discloses method for deleting stale replicas of chunks, the replicas being stored by a plurality of servers, the method comprising:

associating version information with replicas of chunks (column 14, lines 29-37, Mattis);

identifying stale replicas based on the associated version information (column 26, lines 15-21, Mattis);

deleting, by the one of the servers, the one of the replicas that corresponds to one of the stale replicas (column 26, lines 21-22, Mattis);

receiving, from the servers, information concerning replicas stored by the servers (column 8, lines 16-19, Mattis); and

identifying, to one of the servers, one of the replicas that corresponds to one of the deleted stale replicas (column 26, lines 15-21, Mattis).

Regarding Claim 21, Mattis discloses a method wherein the version information for one of the replicas is updated each time a lease is granted for the one of the replicas (column 26, lines 8-15, Mattis).

Regarding Claim 23, Mattis discloses a method wherein the deletion of the stale replicas (column 26, lines 21-22, Mattis) occurs as part of a garbage collection process (Fig.8A, column 21, lines 52-55, Mattis).

Claim Rejections - 35 USC § 103

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

Claim 3 is rejected under 35 U.S.C. 103(a) as being unpatentable over Mattis (US Patent No. 6,209,003) as applied to claims 1,2,4-7,10-13, and 20-25 above, and further in view of Manley (US Patent Publication No. 20030182330).

Regarding Claim 3, Mattis discloses all of the claimed subject matter. However, Mattis does not explicitly disclose a method comprising:

receiving an un-deletion instruction regarding the file; and restoring an original name to the file without permanently deleting the renamed file. On the other hand, Manley discloses a method comprising receiving an un-deletion instruction regarding the file (pg.13, ¶0132, lines 1-8, Manley; wherein the rollback procedure performs the instructions of undoing (which includes un-deleting) changes); and restoring an original name to the file without permanently deleting the renamed file (pg.7, ¶0067, lines 12-28, Manley). It would have been obvious to one of ordinary skill in the art at the time of the invention to incorporate the Manley teachings in the Mattis system. A skilled artisan would have been motivated to combine in order to protect oneself from careless errors. For example, if a user deletes a file by unknowingly, having the option to backtrack and undo the mistake. As a result, this allows the user to double check the systems actions as well as their own.

Claims 8-9 and 14-19 are rejected under 35 U.S.C. 103(a) as being unpatentable over Mattis (US Patent No. 6,209,003) as applied to claims 1,2,4-7,10-13, and 20-25 above, and further in view of "New-Value Logging in the Echo Replicated File System", by Hisgen, Birrell, Jerian, Mann, and Swart, Published 1993; referred to hereinafter as "Hisgen".

Regarding Claim 8; Mattis discloses all of the claimed subject matter. However, Mattis does not explicitly disclose a method comprising:

identifying an orphaned chunk, including:

providing a mapping of file names to chunks, and identifying a chunk, as the orphaned chunk, that is not reachable from any of the file names; and

deleting the orphaned chunk. On the other hand, Hisgen discloses a method comprising identifying an orphaned chunk (pg.24, ¶3, lines 4-5, Hisgen), including: providing a mapping of file names to chunks (pg.23, ¶3, lines 1-3, Hisgen; wherein the Fid map is mapping the file identifiers to the addresses, which examiner interprets as 'chunks'), and identifying a chunk, as the orphaned chunk (pg.24, ¶3, lines 4-5, Hisgen), that are not reachable from any of the file names (pg.24, ¶3, lines 1-4, Hisgen); and deleting the orphaned chunk (pg.24, ¶3, lines 5-8, Hisgen). It would have been obvious to one of ordinary skill in the art at the time of the invention to incorporate the Hisgen teachings in the Mattis system. A skilled artisan would have been motivated to combine in order to recognize and delete information that was no longer accessible within the system. Getting rid of the orphaned files along with the files that are no longer in use would save the system time. Instead of the system performing multiple searches for different files, this permits the system to decrease its workload by gathering all of the useless data at once. As a result, this allows the system to allocate needed space for future use.

Regarding Claims 9 and 15, the combination of Mattis in view of Hisgen, discloses a method wherein metadata is associated with the chunk (column 25, lines 2-8, Mattis); and wherein the deleting the orphaned chunks (pg.24, ¶3, lines 5-8, Hisgen) includes erasing the metadata associated with the orphaned chunk (See Fig.1, pg.28, ¶5, lines 1-4, Hisgen).

Regarding Claims 14,16,18 and 19, the combination of Mattis in view of Hisgen, discloses a method for deleting orphaned chunks of a plurality of chunks stored by a plurality of servers, the method comprising:

providing a mapping of file names to chunks (pg.23, ¶3, lines 1-3, Hisgen);

identifying chunks, as orphaned chunks (pg.24, ¶3, lines 4-5, Hisgen), that are not reachable from any of the file names (pg.24, ¶3, lines 1-4, Hisgen);

deleting, by the one of the servers, the one of the chunks that corresponds to one of the orphaned chunks (pg.24, ¶3, lines 5-8, Hisgen);

receiving, from the servers, information concerning chunks stored by the servers (column 8, lines 16-19, Mattis); and

identifying, to the one of the servers, one of the chunks that corresponds to one of the deleted orphaned chunks (pg.24, ¶3, lines 4-5, Hisgen).

Regarding Claim 17, the combination of Mattis in view of Hisgen, discloses a method wherein the deletion of the orphaned chunks (pg.24, ¶3, lines 5-8, Hisgen) occurs as part of a garbage collection process (Fig. 8A, column 21, lines 52-55, Mattis).

(10) Response to Argument

I. Appellant argues, Mattis does not disclose renaming a file that is identified to be deleted.

Examiner respectfully disagrees. Mattis discloses at column 23, lines 15-23, wherein "*If the fragment is to be deleted, then in step 812 it is deleted from the arena by marking it as deleted and overwriting the data in the fragment. When an object 52 is stored in multiple fragments, and the garbage collection process determines that one of the fragments is to be deleted, then the process deletes all fragments associated with the object*". The 'marking' of the fragment as deleted is a representation of renaming a file and since it is marked as deleted, that therefore makes it the identified file to be deleted. The applicant's specification further supports the representation of the marking of a file as a form of renaming, found at paragraph [0071]; wherein "*Master 130 may, however, actually only rename the file with a deletion timestamp*". Further, column 32, lines 29-37 states "*To accomplish removal of a block found in the cache, however, in step 960 the process sets the deletion flag, and checks the block in with the deletion flag set. As described herein in connection with the check-in process (steps 938 and 944 of FIG. 9B), when the deletion flag is set, the block will be marked as deleted. Thereafter, the block is eventually removed from the Directory Index when the changes reflected in the Open Directory are synchronized to the Directory Index*". The setting of the deletion flag and as the deletion flag is set, the block being marked as deleted, is a further disclosure of renaming the file to be deleted. Lastly, column 3, lines 13-19 recite "*Also, file systems, being designed for user data file management...examples include: support for random access and selective modification, file permissions, support for moving files, support for renaming files ...*". This particular citation was cited for the purpose of revealing that renaming of files is a well-known process within the art for garbage collection.

II. Appellant argues, Mattis does not disclose permanently deleting the renamed file a predetermined amount of time after renaming the identified file as part of a garbage collection process.

Examiner respectfully disagrees. To begin, as background information, well known within the art, garbage collection is a form of automatic memory management. Garbage collection reclaims garbage, or memory used by objects that will never again be accessed or mutated by the application. As such, the Mattis reference is "*a method for garbage collection in a cache of information objects…the garbage collection periodically selects a pool that is storing an amount of data greater than a minimum storage value or high water mark. Each arena in the pool is examiner and selected for garbage collection according to selection criteria. Each fragment within a selected arena is examiner based upon a second set of selection criteria that determine whether the fragment is retained or deleted*" (See ABSTRACT), therefore, the reference as a whole is part of a garbage collection process. Mattis discloses at column 32, lines 29-37 and columns 35-36, lines 5-20, 65-67, and 1-7; respectively, wherein "*To accomplish removal of a block found in the cache, however, in step 960 the process sets the deletion flag, and checks the block in with the deletion flag set. As described herein in connection with the check-in process (steps 938 and 944 of FIG. 9B), when the deletion flag is set, the block will be marked as deleted. Thereafter, the block is eventually removed from the Directory Index when the changes reflected in the Open Directory are synchronized to the Directory Index*" and "*Accordingly, in step 882 the* *process tests whether the matching block has been marked as deleted, and currently has no other processes reading it or writing it. If the values of both the reader counter and the writer counter are zero, then the block has no other processes reading it or writing it...the Open Directory is updated with any changes that were carried out by the process that modified the copy of the block that was obtained in the checkout process. Thereafter, and if the test of step 936 is negative, the process tests whether a delete check-in flag is set. The delete check-in flag indicates that the block is to be deleted after check-in. The delete flag is an argument to the check-in operation. If the flag is set, then in step 944 the process marks the block as deleted. Processing concludes at step 940*". The setting of the deletion flag and in connection with the check-in process, marking the block as deleted, and eventually removing the block fully discloses the permanently deleting the renamed file a predetermined amount of time after renaming the file. Also, because the delete check-in flag indicates that the block is to be deleted after check-in, which is a predetermined amount of time after renaming the file, further discloses the above-argued limitation.

III. Appellant argues, Mattis does not disclose identifying, to one of the servers, one of the chunks that corresponds to the permanently deleted file.

Examiner respectfully disagrees. Mattis discloses at column 23, lines 15-37, wherein "*If the fragment is to be deleted, then in step 812 it is deleted from the arena by marking it as deleted and overwriting the data in the fragment. When an object 52 is stored in multiple fragments, and the garbage collection process determines that one of the fragments is to be deleted, then the process deletes all fragments associated with the object. This may involve following a chain of fragments, of the type shown in FIG. 5, to another arena or even* *another pool...After the fragment is deleted or moved to another arena, in step 814 the Directory Table 110 is updated to reflect the new location of the fragment...When the correct Directory Table block 112a-112n is identified, the disk location value 118 in the block is updated to reflect the new location of the fragment. If the fragment has been deleted, then any corresponding Directory Table entries are deleted*". Since the object is in multiple fragments and the garbage collection process determines that one of the fragments is to be deleted, then all of the fragments associated are affected as well. Also, any corresponding fragments are identified and deleted as well. This information is identified to the server, as understood at column 7, lines 23-29, wherein "*In this context, the term "object" means a network resource or any discrete element of information that is delivered from a server. Examples of objects include Web pages or documents, graphic images, files, text documents, and objects created by Web application programs during execution of the programs, or other elements stored on a server that is accessible through the Internet 20*". Therefore, Mattis fully discloses the argued limitation above.

IV. Appellant argues, since Mattis is completely silent with regard to permanently deleting a renamed file a predetermined amount of time after renaming the file, therefore, Mattis cannot disclose the predetermined amount of time is a user-configurable amount of time.

Examiner respectfully disagrees. To begin, as addressed in the response above (i.e., **II**), Mattis does in fact disclose the limitation of "permanently deleting a renamed file a predetermined amount of time after renaming the file". As such, Mattis discloses at column 22, lines 8-23, wherein "*Preferably, for each pool 200a-200n of a storage device 90a,* *the cache stores or can access a value indicating the amount of disk space in a pool that is currently storing active data. The cache also stores constant "low water mark" and "high water mark" values, as indicated by block 803. When the amount of active storage in a particular pool becomes greater than the "high water mark" value, garbage collection is initiated and carried out repeatedly until the amount of active storage in the pool falls below the "low water mark" value. The "low water mark" value is selected to be greater than zero, and the "high water mark" value is chosen to be approximately 20% less than the total storage capacity of the pool. In this way, garbage collection is carried out at a time before the pool overflows or the capacity of the storage device 90a is exceeded*". The above citation discloses watermarks (both high and low) wherein when an active storage is above the appropriate capacity, then the system is made aware and garbage collection is carried out in order to decrease the capacity back down to the low water mark level. As discussed above, the predetermined amount of time is associated with the permanent deletion of the renamed file during the garbage collection process. As such, the water marks being relied upon have been programmed within the system (by a user) as a way of determining when the storage capacity meets or exceeds the correct amount, making the amount of time user-configurable. A further example is detailed at column 36, lines 1-7, wherein "*Thereafter, and if the test of step 936 is negative, the process tests whether a delete check-in flag is set. The delete check-in flag indicates that the block is to be deleted after check-in. The delete flag is an argument to the check-in operation. If the flag is set, then in step 944 the process marks the block as deleted. Processing concludes at step 940*". As such, because the delete check-in flag indicates that the block is to be deleted after check-in, the program has been programmed or told to do so by the user, thereby making the amount of time user-configurable.

V. Appellant argues, Mattis is completely silent with regard to permanently deleting a renamed file a user-configurable amount of time after renaming the file; therefore, Mattis cannot disclose the user-configurable amount of time differs for different ones of the files.

Examiner respectfully disagrees. As can be seen from the response above (i.e., **IV**), Mattis in fact does disclose the limitation of "permanently deleting a renamed file a user-configurable amount of time after renaming the file". Therefore, as can be seen from the check-in example, columns 35-36, lines 45-48 and lines 1-7, respectively; wherein "*The cache 80 carries out the process of Fig. 9B to check a block into the Open Directory 130 after the block is read, modified, or deleted…if the test of step 936 is negative, the process tests whether a delete check-in flag is set. The delete check-in flag indicates that the block is to be deleted after check-in. The delete flag is an argument to the check-in operation. If the flag is set, then in step 944 the process marks the block as deleted*". As such, once the block has been checked in, and if the block has a delete check-in flag, the system then deletes the block after the check-in. Since the check-in for a file can differ dependent upon the size and amount of information within the block, and also since the information can only be processed one at a time, therefore affects each file differently as far as when the user-configurable amount of time will actually permanently delete the renamed file (This is supported at column 32, lines 29-37 and lines 47-51, wherein "*in connection with the check-in process, when the deletion flag is set, the block is marked as deleted. Thereafter, the block is eventually removed from the Directory Index when the changes reflected in the Open Directory are synchronized to the Directory Index*" and "*Complementary* *checkout check-in processes are used in order to ensure that only one process at a time can modify a Directory Table block, a mechanism that is essential to ensure that the Directory Table always stores accurate information about objects in the cache*").

VI. Appellant argues, Mattis does not disclose identifying a stale chunk based on the versions of the chunks.

Examiner respectfully disagrees. Mattis discloses at column 17, lines 39-46 and column 26, lines 8-22, wherein "*Each pool header 202 stores a Magic number, a Version No. value, a No. of Arenas value, and one or more arena headers 206a-206n. The Version No. value stores a version number of the program or process that created the arenas 206a-206n in the pool. It is used for consistency checks to ensure that the currently executing version of the cache 80 can properly read and write the arenas…If the Read Counter value is high, then the information object has been loaded recently. In that case, in block 1210 the cache sends a positive response message to the requesting process. Otherwise, as indicated in block 1212, the information object has not been loaded recently. Accordingly, as shown in block 1214, the cache sends a negative responsive message to the calling program or process. In block 1216, the cache updates an expiration date value stored in association with the information object to reflect the current date or time. By updating the expiration date, the cache ensures that the garbage collection process will not delete the object, because after the update it is not considered old. In this way, an old object is refreshed in the cache without retrieving the object from its origin, writing it in the cache, and deleting a stale copy of the object*". The citation above discloses each pool header (a pool is a chunk of contiguous disk space, found within a storage device, which can be allocated from pieces of files, or segments of raw disk partitions; See column 17, lines 11-15) containing a version number, which is used for consistency checks to ensure the current version is being executed. If the read counter for the information is high, that indicates is has been recently loaded (i.e., recent version). However, if the information has not been recently loaded, then the block is not a recent version (i.e., stale/old), and a message is sent to the calling program about the old information. The message that is being sent about the stale chunk represents the system identifying the stale chunk. As a result, Mattis discloses the above-argued limitation.

VII. Appellant argues, Mattis does not disclose means for permanently deleting a file during a garbage collection process that occurs after logging deletion of the file and that the examiner did not specifically address this feature, but instead rejected the claim by generally referring to the rejection of claim 1.

Examiner respectfully disagrees. As can be seen from the response above (i.e., **II**), the citations cited of the Mattis reference disclose the process of garbage collection and permanently deleting a file a predetermined amount of time after renaming the identified file. After identifying the file to be deleted and renaming the file (i.e., by either a mark or a flag), the file is stored until its check-in time for deletion occurs. As stated above, column 23, lines 15-23, column 32, lines 29-37, and columns 35-36, lines 5-20, 65-67, and 1-7; respectively. As such, independent claims 1 and 12 were grouped and rejected under the same premise because within independent claim 1, the renaming of the file is inherently logged into the system, thereby including the particular limitation within independent claim 12.

VIII. Appellant argues, Mattis does not disclose a master configured to rename a file that is identified to be deleted, permanently delete one or more chunks associated with the renamed file a predetermined amount of time after renaming the identified file as part of a garbage collection process, and to identify, to one of the servers, one of the chunks that corresponds to one of the one or more permanently deleted chunks.

Examiner respectfully disagrees. To begin, Mattis discloses at column 36, lines 14-21 and 31-67, wherein "*In the preferred embodiment, the methods described herein are carried out using a general-purpose programmable digital computer system of the type illustrated in FIG. 11. Each of the methods can be implemented in several different ways. For example, the methods can be implemented in the form of procedural computer programs, object-oriented programs, processes, applets, etc., in either a single-process or multi-threaded, multi-processing system...FIG. 11 is a block diagram that illustrates a computer system 1100 upon which an embodiment of the invention may be implemented. Computer system 1100 includes a bus 1102 or other communication mechanism for communicating information, and a processor 1104 coupled with bus 1102 for processing information. Computer system 1100 also includes a main memory 1106, such as a random access memory (RAM) or other dynamic storage device, coupled to bus 1102 for storing information and instructions to be executed by processor 1104. Main memory 1106 also may be used for storing temporary variables or other intermediate information during execution of instructions to be executed by processor*

1104...The invention is related to the use of computer system 1100 for caching information objects. According to one embodiment of the invention, caching information objects is provided by computer system 1100 in response to processor 1104 executing one or more sequences of one or more instructions contained in main memory 1106". The computer system referenced throughout the reference corresponds to the 'master', which is configured to rename a file that is identified to be deleted (See response above for **I**), permanently delete one or more chunks associated with the renamed file a predetermined amount of time after renaming the identified file as part of a garbage collection process (See response above for **II** and **III**), and identify, to one of the servers, one of the chunks that corresponds to one of the one or more permanently deleted chunks (See response above for **III**).

IX. Appellant argues, Mattis does not disclose identifying stale replicas based on the associated version information and identifying, to one of the servers, one of the replicas stored by the server that corresponds to one of the deleted stale replicas.

Examiner respectfully disagrees. To begin, it is well known within the art and further described by the Mattis reference at column 1, lines 61-65 and column 8, lines 11-15, wherein "*Each local repository for object replicas is generally referred to as a cache. A client may be able to access replicas from a topologically proximate cache faster than possible from the original web server, while at the same time reducing Internet server traffic...In the preferred embodiment, the cache 80 stores objects on the storage devices 90a-90n. Popular objects are also replicated into a cache. In the preferred embodiment, the cache has finite size, and is stored in main memory or RAM of the proxy 30*". As stated object replicas are referred to as a cache, and therefore, applicant's claimed replicas are discussed and understood within the Mattis reference, since its focus is on the process of garbage collection process in an object cache. Next, Mattis discloses at column 17, lines 39-46 and column 26, lines 8-22, wherein "*Each pool header 202 stores a Magic number, a Version No. value, a No. of Arenas value, and one or more arena headers 206a-206n. The Version No. value stores a version number of the program or process that created the arenas 206a-206n in the pool. It is used for consistency checks to ensure that the currently executing version of the cache 80 can properly read and write the arenas...If the Read Counter value is high, then the information object has been loaded recently. In that case, in block 1210 the cache sends a positive response message to the requesting process. Otherwise, as indicated in block 1212, the information object has not been loaded recently. Accordingly, as shown in block 1214, the cache sends a negative responsive message to the calling program or process. In block 1216, the cache updates an expiration date value stored in association with the information object to reflect the current date or time. By updating the expiration date, the cache ensures that the garbage collection process will not delete the object, because after the update it is not considered old. In this way, an old object is refreshed in the cache without retrieving the object from its origin, writing it in the cache, and deleting a stale copy of the object*". The citation above discloses each pool header (a pool is a chunk of contiguous disk space, found within a storage device, which can be allocated from pieces of files, or segments of raw disk partitions; See column 17, lines 11-15) containing a version number, which is used for consistency checks to ensure the current version is being executed. If the read counter for the information is high, that indicates is has been recently loaded (i.e., recent version). However, if the information has not been recently loaded, then the block is not a recent version (i.e., stale/old), and a message is sent to the calling program about the old information. The message that is being sent about the stale chunk represents the system identifying the stale chunk. As a result, Mattis discloses the above-argued limitation.

X. Appellant argues, Mattis does not disclose version information for one of the replicas that is updated each time a lease is granted for the one of the replicas.

Examiner respectfully disagrees. Mattis discloses at column 26, lines 8-20, wherein "*If the Read Counter value is high, then the information object has been loaded recently. In that case, in block 1210 the cache sends a positive response message to the requesting process. Otherwise, as indicated in block 1212, the information object has not been loaded recently. Accordingly, as shown in block 1214, the cache sends a negative responsive message to the calling program or process. In block 1216, the cache updates an expiration date value stored in association with the information object to reflect the current date or time. By updating the expiration date, the cache ensures that the garbage collection process will not delete the object, because after the update it is not considered old*". Mattis' updating of an expiration date to reflect the current date or time, and by updating the information, the cache ensuring the garbage collection process will not delete the object because it is not old anymore, corresponds to the updating each time a lease is granted.

XI. Appellant argues, neither Mattis nor Manley, disclose restoring an original name to a renamed file without permanently deleting the renamed file.

Examiner respectfully disagrees. Manley discloses at ¶0067, lines 12-28, wherein "*Thus, after a file data block has been modified the snapshot inode 605 contains a pointer to the original inode file system indirect block 510 which, in turn, contains a link to the inode 515. This inode 515 contains pointers to the original file data blocks 520A, 520B and 520C. However, the* *newly written inode 715 includes pointers to unmodified file data blocks 520A and 520B. The inode 715 also contains a pointer to the modified file data block 520C' representing the new arrangement of the active file system. A new file system root inode 705 is established representing the new structure 700. Note that metadata in any snapshotted blocks (e.g. blocks 510, 515 and 520C) protects these blocks from being recycled or overwritten until they are released from all snapshots. Thus, while the active file system root 705 points to new blocks 710, 712, 715 and 520C', the old blocks 510, 515 and 520C are retained until the snapshot is fully released*". The file data block, which contains a pointer to the original inode, allows the system to point to new blocks while still retaining the old (i.e., original) block. Therefore, allowing the original name of the renamed file to be restored. Further, Manley also discloses at ¶0118, wherein "*However, in the illustrative embodiment, if the source inodes received at the destination refer to inodes in the inode map 1400, then the directory stage creates (on the current built-up snapshot directory 1330) a file entry having the desired file name. This name can be exactly the name derived from the source. A hard link 1332 (i.e. a Unix-based link enables multiple names to be assigned to a discrete file) is created between that file on the snapshot directory 1330 and the entry in the purgatory directory. By so linking the entry, it is now pointed to by both the purgatory directory and the file on the snapshot directory itself. When the purgatory directory root is eventually deleted (thereby killing off purgatory) at the end of the data stream transfer, the hard link will remain to the entry, ensuring that the specific entry in the purgatory directory will not be deleted or recycled (given that the entry's link count is still greater than zero) and a path to the data from the file on the new directory is maintained. Every purgatory entry that eventually becomes associated with a file in the newly built tree will be similarly hard linked, and thereby survive deletion of the purgatory directory. Conversely, purgatory entries that are not re-linked will not survive, and are* *effectively deleted permanently when purgatory is deleted*". Since the file that has been marked to be deleted was given a hard link to the original file, represented by the specific entry, so the path to the data remains. Once the file is un-deleted, the file survives deletion and is given its original name because of the original linkage. Lastly, Manley discloses at ¶0132, wherein "*One noted advantage to the rollback according to this embodiment is that it enables the undoing of set of changes to a replicated data set without the need to maintain separate logs or consuming significant system resources. Further the direction of rollback--past-to-present or present-to-past-is largely irrelevant. Furthermore, use of the purgatory directory, and not deleting files, enables the rollback to not affect existing NFS clients. Each NFS client accesses files by means of file handles, containing the inode number and generation of the file. If a system deletes and recreates a file, the file will have a different inode/generation tuple. As such, the NFS client will not be able to access the file without reloading it (it will see a message about a stale file handle). The purgatory directory, however, allows a delay in unlinking files until the end of the transfer. As such, a rollback as described above can resurrect files that have just been moved into purgatory, without the NFS clients taking notice*". As such, because of the undoing of the set of changes and the delay in unlinking files until the end of the transfer, allows the rollback to resurrect files bringing them back to the original status. The combination of Mattis in view of Manley, disclose the above-argued limitation.

XII. Appellant argues, neither Mattis nor Hisgen, disclose identifying, to one of the servers, one of the chunks that corresponds to one of the deleted orphaned

chunks and disclose deleting, by the one of the servers, the one of the chunks that corresponds to one of the orphaned chunks.

Examiner respectfully disagrees. Hisgen discloses at pg.24, ¶3 and ¶4, wherein "*An orphan file is a file that is no longer in the name space, in that no entry in any directory points to it, but it is still open on some client machine and therefore must still be kept in existence. (We use the term "orphan" because an orphan file has no parent directory). The orphan list lists all orphan files*". The list of the orphan files corresponds to identifying of the chunks that correspond to the orphaned chunks. Also, the EchoBox can remove the file from its orphan list and delete the file, which corresponds to the deletion of the orphaned chunks. Therefore, Mattis in view of Hisgen, disclose the above-argued limitations.

XIII. ***Appellant argues Mattis cannot disclose deleting orphaned chunks as part of a garbage collection process.***

Examiner respectfully disagrees. In response to applicant's arguments against the references individually, one cannot show nonobviousness by attacking references individually where the rejections are based on combinations of references. See *In re Keller*, 642 F.2d 413, 208 USPQ 871 (CCPA 1981); *In re Merck & Co.,* 800 F.2d 1091, 231 USPQ 375 (Fed. Cir. 1986). Therefore, Mattis was not relied upon to disclose the deletion of orphaned chunks. The primary reference of Mattis was relied upon for the process of garbage collecting and the Hisgen reference was incorporated to disclose the deletion of the orphaned chunks.

(11) Related Proceeding(s) Appendix

No decision rendered by a court or the Board is identified by the examiner in the Related Appeals and Interferences section of this examiner's answer.

For the above reasons, it is believed that the rejections should be sustained.

An Appeal conference was held on March 15, 2007 with conferees:

Chelcie Daye (Assistant Patent Examiner), Apu Mofiz (SPE), and Tim Vo (SPE)

Respectfully submitted,

CLD

July 18, 2007

Conferees:

Apu Mofiz
Supervisory Patent Examiner [illegible]

Tim Vo
Supervisory Patent Examiner [illegible]

Chelcie Daye
Assistant Patent Examiner

Paul A. Harrity
Attorney for Appellant(s)
Reg. No. 39,574
(571) 432-0800